Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology
Khulna-9203, Bangladesh

Course No: EE 3122
Sessional on Numerical Methods & Statistics

**Experiment No**. 3
**Name of the Experiment**: Find the Roots of Nonlinear Equations

**Objectives**:

[1] To determine the roots of a nonlinear equation of one variable using bisection method
[2] To determine the root of a nonlinear equation using Regula falsi method
[3] To determine the root of a nonlinear equation using Newton's Method
[4] To determine the root of a nonlinear equation using Secant method
[5] To determine the root of a nonlinear equation using Muller's method

**NB:** Follow you class note for manually solving a nonlinear equation. You can also study the materials before writing code. The function of each of the method is written here. Please solve any non-linear equation by using the functions.

## Theory/Introduction:

- Solving nonlinear equations: find $x*$ such that $f(x*) = 0$.
- – Binary search methods: (Bisection, regula falsi)
- – Newton-typed methods: (Newton's method, secant method)
- – Higher order methods: (Muller's method)
- Accelerating convergence: Aitken's Δ2 method

## Bisection Method:

To find the root in [a, b], where $f(a)f(b)<0$.

- Binary search on the given interval [a, b].
  - Suppose f(a) and f(b) have opposite signs.
  - Let $m = (a + b)/2$. Three things could happen for f(m).
    - $f(m) = 0 \Rightarrow$ m is the solution.
    - f(m) has the same sigh as $f(a) \Rightarrow$ solution in [m, b].
    - f(m) has the same sigh as $f(b) \Rightarrow$ solution in [a, m].
- Linear convergence with rate 1/2.

## Pros and Cons

- Pros
  - Easy to implement.
  - Guarantee to converge with guaranteed convergent rate.
  - No derivative required.
  - Cost per iteration (function value evaluation) is very cheap.
- Cons
  - Slow convergence.
  - Do not work for double roots, like solving $(x - 1)^2 = 0$

**Regula falsi (false position)**

- Straight line approximation + intermediate value theorem

- Given two points $(a, f(a)), (b, f(b))$, $a \neq b$, the line equation

$$L(x) = y = f(b) + \frac{f(a) - f(b)}{a - b}(x - b),$$

  and its root, $L(s) = 0$, is $s = b - \frac{a-b}{f(a)-f(b)} f(b)$.

- Use intermediate value theorem to determine $x^* \in [a, s]$ or $x^* \in [s, b]$

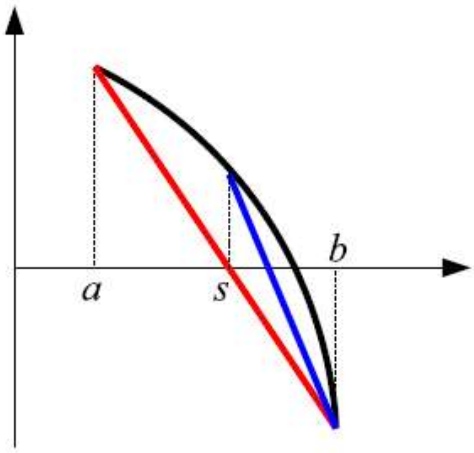
**Convergence of Regula falsi:**

Consider a special case: $(b, f(b))$ is fixed.

- Note $[s, b]$ may not go to zero. (compare to bisection method.)
- Change measurement

$$\frac{|s - x^*|}{|a - x^*|} = \frac{|(b - s) - (b - x^*)|}{|(b - a) - (b - x^*)|}$$

- $b - s = \dfrac{-f(b)}{f(a) - f(b)}(b - a)$.
- Let $m = \dfrac{-f(b)}{f(a) - f(b)} < 1$.




$$\frac{|s - x^*|}{|a - x^*|} = \frac{|m(b - a) - (b - x^*)|}{|(b - a) - (b - x^*)|} < 1$$

- Linear convergence

## Newton's method

- Approximate $f(x)$ by the tangent line $f(x_k) + (x - x_k)f'(x_k)$.

- Find the minimum of the square error

$$\min_x |f(x) - 0|^2 \iff d(f(x))^2/dx = 0$$

- The minimizer is $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
- Convergent conditions
  - $f(x), f'(x), f''(x)$ are continuous near $x^*$, and $f'(x) \neq 0$.
  - $x_0$ is sufficiently close to $x^*$. $\left[\frac{\max|f''|}{2\min|f'|}|x_0 - x^*| < 1\right]$.

## Convergence of Newton's method

- Taylor expansion: for some $\eta$ between $x^*$ and $x_k$

$$f(x^*) = f(x_k) + (x^* - x_k)f'(x_k) + \frac{(x^* - x_k)^2}{2} f''(\eta) = 0$$

$$x^* = x_k - f(x_k)/f'(x_k) - (x^* - x_k)^2 \frac{f''(\eta)}{2f'(x_k)}$$

- Substitute Newton's step $x_k - f(x_k)/f'(x_k) = x_{k+1}$.

$$x^* - x_{k+1} = -(x^* - x_k)^2 \frac{f''(\eta)}{2f'(x_k)}$$

- Quadratic convergence with $\lambda = \left|\frac{f''(x^*)}{2f'(x^*)}\right|$.

## Secant method

- Newton's method requires derivative at each step.
- $f'(x_k)$ can be approximated by $\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}$, which make

$$x_{k+1} = x_k - \frac{x_{k-1} - x_k}{f(x_{k-1}) - f(x_k)} f(x_k).$$

- Convergent conditions
  - $f(x), f'(x), f''(x)$ are continuous near $x^*$, and $f'(x) \neq 0$.
  - Initial guesses $x_0, x_1$ are *sufficiently close* to $x^*$.
    $\max(M|x_0 - x^*|, M|x_1 - x^*|) < 1$, where $M = \max|f''|/2\min|f'|$

## Muller's method

- Approximate $f(x)$ by a parabola.
- A parabola passes $(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))$ is

$$P(x) = f(x_3) + c_2(x - x_3) + d_1(x - x_3)(x - x_2),$$

$$c_1 = \frac{f(x_1) - f(x_3)}{x_1 - x_3}, c_2 = \frac{f(x_2) - f(x_3)}{x_2 - x_3}, d_1 = \frac{c_1 - c_2}{x_1 - x_2}.$$

- We want to find a solution closer to $x_3$. Let $y = x - x_3$ and rewrite $P(x)$ as a function of $y$.

$$\begin{aligned}
P(x) &= f(x_3) + c_2(x - x_3) + d_1(x - x_3)(x - x_2) \\
&= f(x_3) + c_2(x - x_3) + d_1(x - x_3)(x - x_3 + x_3 - x_2) \\
&= f(x_3) + c_2 y + d_1 y(y + x_3 - x_2) \\
&= f(x_3) + (c_2 + d_1(x_3 - x_2))y + d_1 y^2
\end{aligned}$$

- Let $s = c_2 + d_1(x_3 - x_2)$. The solution is

$$y = \frac{-s \pm \sqrt{s^2 - 4d_1 f(x_3)}}{2d_1}, \quad x = x_3 - \frac{s \pm \sqrt{s^2 - 4d_1 f(x_3)}}{2d_1}$$

- Let $x_4$ be the solution closer to $x_3$, $x_4 = x_3 - \frac{s - \text{sign}(s)\sqrt{s^2 - 4d_1 f(x_3)}}{2d_1}$, which equals to (in a more stable way)

$$x_4 = x_3 - \frac{2f(x_3)}{s + \text{sign}(s)\sqrt{s^2 - 4f(x_3)d_1}}.$$

- $x_4$ is the a better approximation to $x^*$ than $x_3$.
- Use $(x_2, f(x_2)), (x_3, f(x_3)), (x_4, f(x_4))$ as next three parameters, and continue the process until converging.

**Summary:**

**Bisection:** To find a root in $[a, b]$, where $f(a)\, f(b) < 0$:
Find $m = (a + b)/2$;
determine whether the root is in $[a, m]$ or in $[m, b]$ by testing whether $f(a)\, f(m) < 0$;
continue the process on an appropriate subinterval.

**Regula falsi and Secant Methods:** Given two estimates of the desired root, $x_a$ and $x_b$, and the corresponding function values, $y_a$ and $y_b$, form

$$x_c = x_b - \frac{x_b - x_a}{y_b - y_a} y_b$$

and proceed as follows:

**Regula Falsi:** If $y_a \cdot y_c < 0$, set $x_b = x_c$ and continue. Otherwise, set $x_a = x_c$ and continue.

**Secant Method:** Set $x_a = x_b, x_b = x_c$ and continue.

**Newton's Method:** Given the current estimate of the root, $x_k$, the value of the function at $x_k$, i.e., $y_k = f(x_k)$, and the value of the derivative at $x_k$, i.e., $y'_k = f'(x_k)$, it follows that

$$x_{k+1} = x_k - \frac{y_k}{y'_k}.$$

**Muller's Method:** Given three points $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$, compute

$$c_1 = \frac{y_2 - y_1}{x_2 - x_1}, \quad c_2 = \frac{y_3 - y_2}{x_3 - x_2}, \quad d_1 = \frac{c_2 - c_1}{x_3 - x_1}, \qquad \text{and } s = c_2 + d_1(x_3 - x_2).$$

The next approximate root is then

$$x = x_3 - \frac{2y_3}{s + \text{sign}(s)\sqrt{s^2 - 4y_3 d_1}}.$$

## MATLAB Code:

### A. Matlab function for bisection

```
function [x, y] = Bisect(fun, a, b, tol, max)
%               Input and output variables
%    fun        string containing name of function
%    [a, b]     interval containing zero
%    tol        allowable tolerance in computed zero
%    max        maximum number of iterations
%    x          vector of approximations to zero
%    y          vector of function values, fun(x)

a(1) = a;       b(1) = b;
ya(1) = feval(fun, a(1));      yb(1) = feval(fun, b(1));
if ya(1) * yb(1) > 0.0
     error('Function has same sign at end points')
end
for i = 1 : max
     x(i) = (a(i) + b(i))/2;  y(i) = feval(fun, x(i));
     if ((x(i)-a(i)) < tol)
          disp('Bisection method has converged'); break;
     end
     if y(i) == 0.0
           disp('exact zero found'); break;
     elseif y(i)*ya(i) < 0
           a(i+1) = a(i); ya(i+1) = ya(i);
           b(i+1) = x(i); yb(i+1) = y(i);
     else
           a(i+1) = x(i); ya(i+1) = y(i);
           b(i+1) = b(i); yb(i+1) = yb(i);
     end;
     iter = i ;
end
if (iter >= max)
     disp('zero not found to desired tolerance');
end
n = length(x); k = 1:n; out = [k'  a(1:n)'  b(1:n)'  x'  y'];
disp('           step           a           b           x           y')
disp(out)
```

## B. Matlab function for regula falsi

```
function [x, y] = Falsi(fun, a, b, tol, max)
%    fun        string containing name of function
%    [a, b]     interval containing zero
%    tol        allowable change in successive iterates
%    max        maximum number of iterations
%    x          vector of approximations to zero
%    y          vector of function values fun(x)

a(1) = a;   b(1) = b;
ya(1) = feval(fun, a(1)); yb(1) = feval(fun, b(1));
if ya(1) * yb(1) > 0.0
     error('Function has same sign at end points')
end
for i = 1 : max
     x(i) = b(i) - yb(i) * (b(i) - a(i))/(yb(i) - ya(i));
     y(i) = feval(fun, x(i));
     if y(i) == 0.0
          disp('exact zero found'); break;
     elseif y(i) * ya(i) < 0
          a(i+1) = a(i); ya(i+1) = ya(i);
          b(i+1) = x(i); yb(i+1) = y(i);
     else
          a(i+1) = x(i); ya(i+1) = y(i);
          b(i+1) = b(i); yb(i+1) = yb(i);
     end;
     if ((i>1) & (abs(x(i)-x(i-1)) < tol))
          disp('Falsi method has converged'); break;
     end
     iter = i ;
end
if (iter >= max)
     disp('zero not found to desired tolerance');
end
n = length(x);  k = 1:n;  out = [k'  a(1:n)'  b(1:n)'  x'  y'];
disp('              step      a          b          x          y')
disp(out)
```

## C. Matlab function for Secant method

```
function [x, y] = Secant(fun, a, b, tol, max)
% Find a zero using secant method.
%       fun         string containing name of function
%       a, b        first two estimates of the zero
%       tol         tolerance for change in computed zero

%       max         maximum number of iterations
%       x           vector of approximations to zero
%       y           vector of function values fun(x)
x(1) = a;                          x(2) = b;
y(1) = feval(fun, x(1));      y(2) = feval(fun, x(2));
for i = 2 : max
     x(i+1) = x(i) - y(i) * (x(i) - x(i-1))/(y(i) - y(i-1));
     y(i+1) = feval(fun, x(i+1));
     if (abs(x(i+1)-x(i)) < tol)
          disp('method has converged'); break;
     end
     if y(i) == 0.0
          disp('exact zero found'); break;
     end
     iter = i;
end
if (iter >= max)
     disp('zero not found to desired tolerance');
end
n = length(x);      k = 1:n;      out = [k'   x'   y'];
disp('            step            x            y'), disp(out)
```

## D. Matlab function for Newton's method

```
function [x, y] = Newton(fun, fun_pr, x1, tol, max)
% Find zero near x1 using Newton's method.
% Input :
%       fun        string containing name of function
%       fun_pr     name of derivative of function
%       x1         starting estimate
%       tol        allowable tolerance in computed zero
%       max        maximum number of iterations
% Output :
%       x          (row) vector of approximations to zero
%       y          (row) vector fun(x)

x(1) = x1;
y(1) = feval(fun, x(1));
y_pr(1) = feval(fun_pr, x(1));
for i = 2 : max
    x(i) = x(i-1) - y(i-1)/y_pr(i-1);
    y(i) = feval(fun, x(i));
    if abs(x(i) - x(i-1)) > tol
        disp('Newton method has converged'); break;
    end
    y_pr(i) = feval(fun_pr, x(i));
    iter = i;
end
if (iter >= max)
    disp('zero not found to desired tolerance');
end
n = length(x);     k = 1:n;  out = [k'  x'  y'];
disp('           step         x          y')
disp(out)
```

## Experiment:

For the problem, find the positive real zero of the function that follow; find consecutive integers "*a*" and "*b*" that bracket the root to use as starting values for the bisection, regula falsi, or secant method. Use (*a*+*b*)/2 as the starting value of Newton's method and as the starting value for Muller's method. Find the zero using bisection, regula falsi, secant method, Newtons method, and Muller's method. Problem: $f(x)=x^2 - 5$,c $f(x)= x^3 - 6$

## Report:

i. Perform the manual calculation of the supplied problem by using different methods as well as perform by using Matlab/C/C++